PATENT APPLICATION
EXPRESS MAIL No.EL665602995US
ATTORNEY DOCKET No.CIRC015
CLIENT/MATTER NO. 80142.2005

SYSTEM AND METHOD FOR SHIFTING FUNCTIONALITY BETWEEN MULTIPLE WEB SERVERS

BACKGROUND OF THE INVENTION

1. Related Applications.

The present invention claims priority from U.S. Provisional Patent Application No. 60/197,490 entitled CONDUCTOR GATEWAY filed on April 17, 2000.

2. Field of the Invention.

The present invention relates, in general, to network information access and, more particularly, to software, systems and methods for shifting functionality between multiple web servers in a coordinated multi-server web site.

3. Relevant Background.

Increasingly, business data processing systems, entertainment systems, and personal communications systems are implemented by computers across networks that are interconnected by internetworks (e.g., the Internet). The Internet is rapidly emerging as the preferred system for distributing and exchanging data. Data exchanges support applications including electronic commerce, broadcast and multicast messaging, videoconferencing, gaming, and the like.

The Internet is a collection of disparate computers and networks coupled together by a web of interconnections

using standardized communications protocols. The Internet is characterized by its vast reach as a result of its wide and increasing availability and easy access protocols. Unfortunately, the ubiquitous nature of the Internet results in variable bandwidth and quality of service between points. The latency and reliability of data transport is largely determined by the total amount of traffic on the Internet and so varies wildly seasonally and throughout the day. Other factors that affect quality of service include equipment outages and line degradation that force packets to be rerouted, damaged and/or dropped. Also, routing software and hardware limitations within the Internet infrastructure may create bandwidth bottlenecks even when the mechanisms are operating within specifications.

Often, a given entity will provide multiple services using multiple servers. For example, a typical suite of Internet services might include a web server, a mail server, a file transfer server, a chat server and the like. These servers may execute on a single machine or on multiple machines. Characteristically, the services are provided on a well-defined machine at a known network address so that users can readily find the machine that is needed to perform a particular function. As a result, the functionality is bound to particular servers. More recently, efforts have been made to replicate services on geographically or topologically distributed servers to improve capacity and performance. However, these solutions maintain the static binding between functionality and particular hardware/software platforms that implement the functionality.

Unfortunately, the ubiquitous nature of the Internet results in variable bandwidth and quality of service

between points. The latency and reliability of data transport is largely determined by the total amount of traffic on the Internet and so varies wildly seasonally and throughout the day. Other factors that affect quality of service include equipment outages and line degradation that force packets to be rerouted, damaged and/or dropped. Also, routing software and hardware limitations within the Internet infrastructure may create bandwidth bottlenecks even when the mechanisms are operating within specifications.

As a result of this static allocation of functionality, the functionality provided by the network-connected servers behaves unpredictably, and may give less than optimal performance. Even where functionality is replicated across multiple servers, service remains sub-optimal. Moreover, load balancing across multiple servers providing replicated functionality is problematic and often leads to inefficient use of some resources while other resources are overburdened.

A particular need exists in environments that involve multiple users accessing a network resource such as a web server. Web servers are typically implemented with rich functionality and are often extensible in that the functionality provided can be increased modularly to provide general-purpose and special-purpose functions. Examples include information services, broadcast, multicast and video conference services, as well as most electronic commerce (e-commerce) applications. In these applications it is important to that functionality provided by network-connected resources be provided in a dependable, timely and efficient manner.

In e-commerce applications, it is important to provide a satisfying buyer experience that leads to a

purchase transaction. To provide this high level of service, a web site operator must ensure that data is exchanged with the customer in the most usable and efficient fashion. An e-commerce transaction may involve several services including search services, file retrieval services, shopping cart services and payment services. These varied services are typically delivered from a single network node, however the node may be less than optimal for a particular customer. Even if multiple server nodes were available, current technology does not provide a way to select a particular network node. Moreover, there is no mechanism for selectively shifting particular functionality among available serving nodes to improve performance on a function-by-function basis.

Until now, however, the e-commerce site owner has had little or no control over the transport mechanisms through the Internet that affect the latency and quality of service. This is akin to a retailer being forced to deal with a customer by shouting across the street, never certain how often what was said must be repeated, and knowing that during rush hour communication would be nearly impossible. While efforts are continually being made to increase the capacity and quality of service afforded by the Internet, it is contemplated that congestion will always impact the ability to predictably and reliably offer a specified level of service. Moreover, the change in the demand for bandwidth increases at a greater rate than does the change in bandwidth supply, ensuring that congestion will continue to be an issue into the foreseeable future. A need exists for a system to exchange data over the Internet that provides a high quality of service even during periods of congestion.

Many electronic commerce transactions are abandoned by the user because system performance degradations frustrate the purchaser before the transaction is consummated. While a transaction that is abandoned while a customer is merely browsing through a catalog may be tolerable, abandonment when the customer is just a few clicks away from a purchase is highly undesirable. However, existing Internet transport mechanisms and systems do not allow the e-commerce site owner any ability to distinguish between the "just browsing" and the "about to buy" customers. In fact, the vagaries of the Internet may lead to the casual browser receiving a higher quality of service while the about-to-buy customer becomes frustrated and abandons the transaction.

Partial solutions have been implemented by systems that cache Internet content at multiple distributed locations. In theory, when content can be served from a cache location, it can be delivered with lower latency than if it were served from a single originating web server. However, the content must be copied from its origin to the multiple caches resulting in a tremendous volume of data that must be duplicated and transported. Moreover, it is difficult to keep all of the cache copies coherent with the origin. Furthermore, a cache handles only static content and does not distribute functionality to locations from which it can be more efficiently delivered. Hence, caching is only a partial solution to the many web sites that use dynamically generated web pages.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a system for providing functionality and services from a

plurality of network-connected resources where the mechanisms, such as computer code modules, implementing the functionality are dynamically assigned to one or more of the various network connected resources. Users are dynamically and preferably transparently directed to the network resource in which requested functionality currently resides.

A system for providing network resources from a plurality of network-connected resources to at least one network-connected client computer. A management component is coupled to each of the resources. A shifting component within the management component operates to shift data and program components between the network-connected resources so as to configure a selected resource to implement a specified set of functionality and/or provide a specified set of resources. A redirection component responsive to a client request for the specified set of functionality redirects the requesting client to a server offering the specified functionality.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

FIG. 2a-FIG. 2d show in block-diagram form entity relationships in a system in accordance with the present invention;

FIG. 3 illustrates in block-diagram form significant components of a particular implementation of a system in accordance with the present invention;

FIG. 4 shows a domain name system used in an implementation of the present invention;

FIG. 5 shows front-end components of FIG. 2 in greater detail;

FIG. 6 shows back-end components of FIG. 2 in greater detail; and

FIG. 7 shows a conceptual block diagram of the system of FIG. 2 in an alternative context.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In establishing a network architecture for providing services, there is a continuous compromise made in selecting where, topologically, to locate the hardware and software that provides the services. In almost all networks, quality of service amongst the nodes using the services will differ. This compromise is more apparent in Internet-based systems because the differential between users is greater, is unpredictable, and varies significantly over time. Often, the architect has incomplete knowledge of where the nodes that will access the services exist. The present invention addresses this compromise by providing a generic software/hardware infrastructure in which the functionality is provided by any or all of a large number of nodes. Functionality is dynamically shifted amongst the available nodes to provide improved service to all participants.

The present invention generally involves systems, methods and software for delivering functionality from network-connected resources (e.g., servers) to network-connected appliances (e.g., clients). It is recognized that the roles of client and server are flexibly defined in an operating network as some entities play both roles at any given time and may in fact play both roles at the same time. These terms are used for convenience herein

for ease of explanation, however, unless expressly indicated otherwise the terminology is not to be considered a limitation on the broader teachings of the present invention as set forth in the claims. Further, the examples are directed at web-based systems using the Internet, however, more generally, the present invention relates to any type of functionality or resources provided over a network and need not be implemented as a web-based system. Hence, unless specified otherwise, a "web server" is synonymous with a "server" or other entity that delivers functionality in response to requests. Likewise, a "browser" is synonymous with a client or other entity that requests and uses a particular set of functionality.

The particular examples herein often refer to a system in which a particular set of functionality is embodied in a specific web site controlled by an entity such as a business or service provider. This model fits conventional Internet services deployment in which functionality is statically bound to one or more specific network nodes. In one respect, the present invention can be viewed as an augmentation of this conventional model that enables functionality to be distributed from its statically assigned location to a variety of dynamically assigned nodes that implement the web site's functionality. In a second aspect, the present invention contemplates that the entity-controlled web site may exist only virtually (e.g., none of the web site functionality is statically bound to a particular node) and that all of the site functionality is embodied in the nodes to which the functionality is distributed.

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication

channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments, including wireless network environments, as well as small network environments such as conventional LAN systems.

The present invention is particularly useful in applications where there is an large amount of data communicated between web servers and web clients (i.e., browser software) or where timeliness (e.g., low latency transport) is important. For example, real-time stock quotes, multi-player games, multi-tiered service to ASP (application service provider) software distribution models benefit from the improvements provided by the present invention. Although the present invention will be described in terms of particular applications, these examples are provided to enhance understanding and are not a limitation of the essential teachings of the present invention.

The current experience of using an overloaded, topologically distant or unavailable web site that is one of frustration and constant and unexplained errors. In terms of conventional phone-order shopping, this is akin to a customer phoning a favorite mail order company only to receive a constant busy signal or non-stop ringing or to be connected to an operator only to be disconnected prior to the completion of the order placement. Under the present invention, web site operators have a mechanism to communicate with their customers even if they cannot handle their web query immediately. One aspect of the present invention is an ability to serve alternative

content and/or functionality in response to a request for particular content and/or functionality. For example, the invention contemplates a system that presents the user with a pre-defined set of web pages which can apologize for delays, instruct the user on what to do in case of an emergency, advertise additional options or new product offerings, and/or provide other functionality such as call-center type features including queue management.

The present invention uses a web site delivery system in which a plurality of front-end web servers and back-end web servers cooperate to deliver content and services of the web site. A front-end web server is an application program that is enabled to generate web pages (e.g., HTML pages) in response to HTTP requests received from a client software application such as a web browser. A back-end or "originating" web server is a software application that receives requests from the front-end web server and is logically close to one or more network resources (e.g., data storage, database servers, and the like). The term "logically close" as used herein refers to quality of service between two points and may reflect physical proximity, topological proximity or other factors that affect quality of service and speed of service between two points.

A web site is conventionally implemented by including all the functionality implemented by the web site in a web server or web servers that are logically close to the network resources to be served by that web site. In contrast, the present invention implements web site functionality and behavior within one or more front-end web server(s) that communicate over the public network or WAN to obtain the network resources or possess the network resources locally. Preferably, the front-end web server

includes an enhanced communication channel for communicating with the back-end web server. This shift in functionality from the back-end to the front-end provides improved performance in many applications. CPU intensive processing can be performed by the front-end server thereby relieving the back-end web server to handle requests. Also, because multiple front-ends may access the same back-end servers, the CPU intensive functionality is distributed over a larger number of machines as needed to provide improved performance and access to a larger number of clients.

For purposes of this document, a web server is a computer running server software coupled to the World Wide Web (i.e., "the web") that delivers or serves web pages. The web server has a unique IP address and accepts connections in order to service requests by sending back responses. A web server differs from a proxy server or a gateway server in that a web server has resident a set of resources (i.e., software programs, data storage capacity, and/or hardware) that enable it to execute programs to provide an extensible range of functionality such as generating web pages, accessing remote network resources, analyzing contents of packets, reformatting request/response traffic and the like using the resident resources. In contrast, a proxy simply forwards request/response traffic on behalf of a client to resources that reside elsewhere, or obtains resources from a local cache if implemented. A web server in accordance with the present invention may reference external resources of the same or different type as the services requested by a user, and reformat and augment what is provided by the external resources in its response to the user. Commercially available web server software includes Microsoft Internet Information Server (IIS), Netscape

Netsite, Apache, among others. Alternatively, a web site may be implemented with custom or semi-custom software that supports HTTP traffic.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented. Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, or peer-to-peer type network. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another network mechanism such as a fibre channel fabric or conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Micrsosoft Windows, Macintosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, and the like. Moreover, the present invention is readily extended to network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an

individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

FIG. 2a-FIG. 2d illustrate entity relationships involved in an interaction in which functionality provided by an originating web site 210 is provided from a variety of sources including front-ends 201. A single client 205 is shown in FIG. 2a-FIG. 2d, however, a number of clients will be enabled to use the system simultaneously. Each client 205 has a network connection that supports a communication channel to one or more originating servers 210 in which a particular web site is implemented.

In a typical client-server transaction shown in FIG. 2a, a client makes a request for resources and/or functionality implemented on originating server 210. In a web-based implementation, this request takes the form of an HTTP request datagram specifying a uniform resource locator (URL) that identifies originating server 210. As in conventional networks, various routing actions are taken within network 101 (described below) to direct the request datagram to its intended destination. In a conventional environment, client 205 waits for a response to the request datagram.

Management component 207 shown in FIG. 2c, in practice, embodies a number of routing and control functions that may be implemented in a number of servers as opposed to the single entity shown in FIG. 2b-FIG. 2c. Mechanisms shown in Fig. 4 are used to redirect client 205 to one of a plurality of front-end servers 201 shown in FIG. 2c. Another function of management component 207 is to dynamically maintain and shift content and functionality between originating server 210 and the various available front-end machines 201. In a particular

implementation, management component 207 maintains an out-of-band or in-band connection 202 with each front-end server 201 and each originating server 210. Connection 202 enables the dynamic shifting of functionality as needed to meet the needs of a particular application.

Unlike conventional networks, the present invention operates as shown in FIG. 2b to route the request from client 205 to a particular front-end 201 that is able to respond to the request from client 205 as shown in FIG. 2d. Non-selected front-ends 201, shown in phantom in FIG. 2d, are not involved in responding to the request. Preferably, client 205 is unaware that content and/or functionality is being provided by an entity other than originating server 210. In fact, it is contemplated that originating server 210 may not exist in a physical sense at all. So long as the desired functionality is provided by the available front-ends 201, originating server 210 may become unnecessary as a physical entity.

Management component 207 may shift functionality after the request is received, or proactively before the request is received based on predictive logic. Front-end servers may cache or locally store content and program code that are frequently used to control delays in porting functionality between nodes. It is contemplated that management component may select originating server 210 itself in some circumstances to provide content and functionality that is most efficiently deployed at a particular time or in particular circumstances by server 210. In this manner, functionality, content, and behavior is dynamically shifted amongst available resources.

In a particular example, management component 207 measures and maintains metrics associated with each front-end server 201 over channel 202. These metrics include,

for example, processor load, quality of service between various points, memory resource load, and other measurements that impact the ability of a front-end server to efficiently respond to requests. Management component 207 uses these metrics in the selection of a particular front-end 201 to respond to a particular request. Preferably, management component 207 also maintains an inventory of the content and functionality available at each front-end server 201 so that requests can be directed to a front-end that is already configured to provide the desired functionality.

In the preferred web-based implementation, front-end servers 201 are implemented by web server software that supports dynamic expansion of functionality. Several commercial off-the-shelf web server packages offer such extensible functionality through the use of plug-ins, servelets, agents and other modular program components. In some cases, it is contemplated that special purpose web server software will be used in front-ends 201 that implement more streamlined and compact software than is possible with off-the-shelf general purpose products. It is contemplated that any computer upon which front-end 201 executes will support multiple instances of front-end 201. In other words, a network node upon which a front-end 201 resides may actually have multiple front-ends 201 in existence at any given time. In this specific example, client 205 comprises a network-enabled graphical user interface such as a World Wide Web (web) browser. However, the present invention is readily extended to client software other than conventional web browser software. Any client application that can access a standard or proprietary user level protocol or language for network access is a suitable equivalent.

For convenience, the term "web site" is used interchangeably with "web server" in the description herein although it should be understood that a web site comprises a collection of content, programs and processes implemented on one or more web servers. A web site is owned by the content provider such as an e-commerce vendor whereas a web server refers to set of programs running on one or more machines coupled to an Internet, intranet, or other network node. The web site 210 may be hosted on the site owner's own web server, or hosted on a web server owned by a third party. A web hosting center is an entity that implements one or more web sites on one or more web servers using shared hardware and software resources across the multiple web sites. In a typical web infrastructure, there are many web browsers, each of which has a TCP connection to the web server in which a particular web site is implemented. The present invention adds two components to the infrastructure: a front-end 201 and back-end 203. Front-end 201 and back-end 203 are coupled by a managed data communication link 202 that forms, in essence, a private network.

FIG. 3 shows a particular implementation of the functionality shifting system of FIG. 2a-FIG. 2d. that uses not only front-end servers 201, but also one or more back-end servers 203 to further improve functionality shifting performance. In the particular implementation shown in FIG. 3, functionality can be served from originating server 210, back-end server 203 and/or front-end servers 201 as needed to meet the needs of a particular application.

In FIG. 3, a private network 200 implemented within the Internet infrastructure. Private network 200 expedites and prioritizes communications between a client

205 and a web site 210. In the specific examples herein client 205 comprises a network-enabled graphical user interface such as a web browser. However, the present invention is readily extended to client software other than conventional web browser software. Any client application that can access a standard or proprietary user level protocol for network access is a suitable equivalent. Examples include client applications for file transfer protocol (FTP) services, voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose Internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may access a network application such as a database management system (DBMS) in which case the client application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client application may communicate via a wireless application protocol (WAP) or the like.

Front-end mechanism 201 serves as an access point for client-side communications. Front-end 201 implements a gateway that, from the perspective of client 205, appears to be the web site 210. Front-end 201 comprises, for example, a computer that sits "close" to clients 205. By "close", it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and a web site 210. Desirably, the connection between front-end computers and clients 205 have low latency and high bandwidth. For example, the fastest available connection may be implemented in point of presence (POP) of an Internet service provider (ISP) 106 used by a particular client

205. However, the placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications, it may be more practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance between front-end 201 and clients 205 may be desirable in some applications as this distance will allow for selection amongst a greater number front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when particular routes and/or front-ends become congested or otherwise unavailable.

Enhanced communication channel 212 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from originating server 210. In a particular example, transport mechanism 212 communicates data packets using a proprietary protocol over the Internet. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general-purpose network 101. Unlike the general purpose Internet, front-end 201 and back-end 203 may be assigned to serve accesses to one or more particular originating servers 210 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the web site owner. The present invention implements a many-to-many mapping of front-ends to back-ends. Because the front-end to back-end mappings can by dynamically changed, a fixed hardware infrastructure can be logically reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

Front-end 201 together with back-end 203 function to reduce traffic across the transport morphing protocol TMP™ link 212 and to improve response time for selected browsers. Traffic across the TMP link 212 is reduced by compressing data and serving browser requests from cache for fast retrieval. Also, the blending of request datagrams results in significantly fewer request:acknowledge pairs across the TMP link 212 as compared to the number required to reliably send the packets individually between front-end 201 and back-end 203. This action reduces the overhead associated with transporting a given amount of data, although conventional request:acknowledge traffic is still performed on the links coupling the front-end 201 to client 205 and back-end 203 to a web server. Moreover, resend traffic is significantly reduced further reducing the traffic. Response time is further improved for select privileged users and for specially marked resources by determining the priority for each HTTP transmission.

In one embodiment, front-end 201 and back-end 203 are closely coupled to the Internet backbone. This means they have high bandwidth connections, can expect fewer hops, and have more predictable packet transit time than could be expected from a general-purpose connection. Although it is preferable to have low latency connections between front-ends 201 and back-ends 203, a particular strength of the present invention is its ability to deal with latency by enabling efficient transport and traffic prioritization. Hence, in other embodiments front-end 201 and/or back-end 203 may be located farther from the Internet backbone and closer to clients 205 and/or web servers 210. Such an implementation reduces the number of hops required to reach a front-end 201 while increasing the number of hops within the TMP link 212 thereby

yielding control over more of the transport path to the management mechanisms of the present invention.

Clients 205 no longer conduct all data transactions directly with the web server 210. Instead, clients 205 conduct some and preferably a majority of transactions with front-ends 201, which simulate the functions of web server 210. Client data is then sent, using TMP link 212, to the back-end 203 and then to the web server 210. Running multiple clients 205 over one large connection provides several advantages:

- Since all client data is mixed, each client can be assigned a priority. Higher priority clients, or clients requesting higher priority data, can be given preferential access to network resources so they receive access to the channel sooner while ensuring low-priority clients receive sufficient service to meet their needs.

- The large connection between a front-end 201 and back-end 203 can be permanently maintained, shortening the many TCP/IP connection sequences normally required for many clients connecting and disconnecting.

Using a proprietary protocol allows the use of more effective techniques to improve data throughput and permits better use of existing bandwidth during periods when the network is congested.

A particular advantage of the architectures shown in FIG. 2a-FIG. 2d and FIG. 3 is that it is readily scaled. Any number of client machines 205 may be supported. In a similar manner, a web site owner may choose to implement a site using multiple web servers 210 that are co-located or

distributed throughout network 101. To avoid congestion, additional front-ends 201 may be implemented or assigned to particular web sites. Each front-end 201 is dynamically re-configurable by updating address parameters to serve particular web sites. Client traffic is dynamically directed to available front-ends 201 to provide load balancing. Hence, when quality of service drops because of a large number of client accesses, an additional front-end 201 can be assigned to the web site and subsequent client requests directed to the newly assigned front-end 201 to distribute traffic across a broader base.

In the particular examples, this is implemented by a front-end manager component 217 that communicates with multiple front-ends 201 to provide administrative and configuration information to front-ends 201. Each front-end 201 includes data structures for storing the configuration information, including information identifying the back-ends 203 of web servers 210 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 may include information for prioritizing specific data from and to particular clients, quality of service information, and the like.

Similarly, additional back-ends 203 can be assigned to a web site to handle increased traffic. Back-end manager component 219 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold current configuration state, quality of service information and the like. In the particular examples, front-end manager 217 and back-end manager 219 serve multiple web sites 210 and so are able to manipulate the

number of front-ends and back-ends assigned to each web site 210 by updating this configuration information. When the congestion for the site subsides, the front-ends 201 and back-ends 203 can be reassigned to other, busier web sites. These and similar modifications are equivalent to the specific examples illustrated herein.

In the case of web-based environments, front-end 201 is implemented using custom or off-the-shelf web server software. Front-end 201 is readily extended to support other, non-web-based protocols, however, and may support multiple protocols for varieties of client traffic. Front-end 201 processes the data traffic it receives, regardless of the protocol of that traffic, to a form suitable for transport by TMP 212 to a back-end 203. Hence, most of the functionality implemented by front-end 201 is independent of the protocol or format of the data received from a client 205. Hence, although the discussion of the exemplary embodiments herein relates primarily to front-end 201 implemented as a web server, it should be noted that, unless specified to the contrary, web-based traffic management and protocols are merely examples and not a limitation of the present invention.

As shown in FIG. 2, in accordance with the present invention a web site is implemented using an originating web server 210 operating cooperatively with the web server of front-end 201. More generally, any network service (e.g., FTP, VoIP, NNTP, MIME, SMTP, Telnet, DBMS) can be implemented using a combination of an originating server working cooperatively with a front-end 201 configured to provide a suitable interface (e.g., FTP , VoIP, NNTP, MIME, SMTP, Telnet, DBMS, WAP) for the desired service. In contrast to a simple front-end cache or proxy software, implementing a server in front-end 201 enables portions of

the web site (or other network service) to actually be implemented in and served from both locations. The actual web pages or service being delivered comprises a composite of the portions generated at each server. Significantly, however, the web server in front-end 201 is close to the browser in a client 205 whereas the originating web server is close to all resources available at the web hosting center at which web site 210 is implemented. In essence the web site 210 is implemented by a tiered set of web servers comprising a front-end server 201 standing in front of an originating web server.

This difference enables the web site functionality to be implemented and distributed so as to take advantage of the unique position each web server has with respect to the client 205. By way of a particular example, assume an environment in which the front-end web server 201 is located in the ISP's location for a particular set of clients 205. In such an environment, clients 205 can access the front-end web server 205 without actually traversing the network 101. Hence, network delays and variability are substantially eliminated.

In order for a client 205 to obtain service from a front-end 201, it must first be directed to a front-end 201 that can provide the desired service. FIG. 4 illustrates a domain name server (DNS) redirection mechanism that illustrates an example of how a client 205 is connected to a particular front-end 201. Preferably, client 205 does not need to be aware of the location of front-end 201, and initiates all transactions as if it were contacting the originating server 210. The public DNS system is defined in a variety of Internet Engineering Task Force (IETF) documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In a

typical environment, a client 205 executes a browser 301, TCP/IP stack 303, and a resolver 305. For reasons of performance and packaging, browser 301, TCP/IP stack 303 and resolver 305 are often grouped together as routines within a single software product.

Browser 301 functions as a graphical user interface to implement user input/output (I/O) through monitor 311 and associated keyboard, mouse, or other user input device (not shown). Browser 301 is usually used as an interface for web-based applications, but may also be used as an interface for other applications such as email and network news, as well as special-purpose applications such as database access, telephony, and the like. Alternatively, a special-purpose user interface may be substituted for the more general-purpose browser 301 to handle a particular application.

TCP/IP stack 303 communicates with browser 301 to convert data between formats suitable for browser 301 and IP format suitable for Internet traffic. TCP/IP stack also implements a TCP protocol that manages transmission of packets between client 205 and an Internet service provider (ISP) or equivalent access point. IP protocol requires that each data packet include, among other things, an IP address identifying a destination node. In current implementations the IP address comprises a 32-bit value that identifies a particular Internet node. Non-IP networks have similar node addressing mechanisms. To provide a more user-friendly addressing system, the Internet implements a system of domain name servers that map alpha-numeric domain names to specific IP addresses. This system enables a name space that is more consistent reference between nodes on the Internet and avoids the need for users to know network identifiers, addresses,

routes and similar information in order to make a connection.

The domain name service is implemented as a distributed database managed by domain name servers (DNSs) 307 such as DNS_A, DNS_B and DNS_C shown in FIG. 3. Each DNS relies on <domain name:IP> address mapping data stored in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files typically comprise text files that are read by a local name server, and hence become available through the name servers 307 to users of the domain system.

The user programs (e.g., clients 205) access name servers through standard programs such as resolver 305. Resolver 305 includes an address of a DNS 307 that serves as a primary name server. When presented with a reference to a domain name (e.g., http://www.circadence.com), resolver 305 sends a request to the primary DNS (e.g., DNS_A in FIG. 3). The primary DNS 307 returns either the IP address mapped to that domain name, a reference to another DNS 307 which has the mapping information (e.g., DNS_B in FIG. 3), or a partial IP address together with a reference to another DNS that has more IP address information. Any number of DNS-to-DNS references may be required to completely determine the IP address mapping.

In this manner, the resolver 305 becomes aware of the IP address mapping which is supplied to TCP/IP component 303. Client 205 may cache the IP address mapping for future use. TCP/IP component 303 uses the mapping to supply the correct IP address in packets directed to a particular domain name so that reference to the DNS system need only occur once.

In accordance with the present invention, at least one DNS server 307 is owned and controlled by system components of the present invention. When a user accesses a network resource (e.g., a web site), browser 301 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210 and server 210 performs a redirection of the browser to the system owned DNS server (i.e., DNC_C in FIG. 3). In a second embodiment, domain:address mappings within the DNS system are modified such that resolution of the of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of further redirecting the browser 301 to the best available front-end 201.

Unlike a conventional DNS server, however, the system-owned DNS_C in FIG. 3 receives domain:address mapping information from a redirector component 309. Redirector 309 is in communication with front-end manager 217 and back-end manager 219 to obtain information on current front-end and back-end assignments to a particular server 210. A conventional DNS is intended to be updated infrequently by reference to its associated master file. In contrast, the master file associated with DNS_C is dynamically updated by redirector 309 to reflect current assignment of front-end 201 and back-end 203. In operation, a reference to web server 210 (e.g., http://www.circadence.com) may result in an IP address returned from DNS_C that points to any selected front-end 201 that is currently assigned to web site 210. Likewise,

web site 210 may identify a currently assigned back-end 203 by direct or indirect reference to DNS_C.

Front-end 201 typically receives information directly from front-end manager 217 about the address of currently assigned back-ends 203. Similarly, back-end 203 is aware of the address of a front-end 201 associated with each data packet. Hence, reference to the domain system is not required to map a front-end 201 to its appropriate back-end 203.

FIG. 5 illustrates principle functional components of an exemplary front-end 201 in greater detail. Primary functions of the front-end 201 include translating transport control protocol (TCP) packets from client 205 into TMP packets used in the system in accordance with the present invention. Transport morphing protocol™ and TMP™ are trademarks or registered trademarks of Circadence Corporation in the United States and other countries. It is contemplated that the various functions described in reference to the specific examples may be implemented using a variety of data structures and programs operating at any location in a distributed network. For example, a front-end 201 may be operated on a network appliance 107 or server within a particular network 102, 103, or 104 shown in FIG. 1. The present invention is readily adapted to any application where multiple clients are coupling to a centralized resource. Moreover, other transport protocols may be used, including public and proprietary transport protocols.

TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792, RFC1112 that are incorporated by reference herein. For

ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement file transfer protocol (FTP), mail protocols such as simple mail transport protocol (SMTP), structured query language (SQL) and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

To improve performance, front-end 201 optionally includes a caching mechanism 403. Cache 403 may be

implemented as a passive cache that stores frequently and/or recently accessed web pages or as an active cache that stores network resources that are anticipated to be accessed. In non-web applications, cache 403 may be used to store any form of data representing database contents, files, program code, and other information. Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for data within cache 403. If the request can be satisfied from cache 403, the data is supplied directly without reference to web server 210 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by web sever 210. Also, cache 403 may proactively update the cache contents even before a request is received for particularly important or frequently used data from web server 210. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out, or random eviction. When the requested data is not within cache 403, a request is processed to web server 210, and the returned data may be stored in cache 403.

Several types of packets will cause parser 402 to forward a request towards web server 210. For example, a request for data that is not within cache 403 (or if optional cache 403 is not implemented) may require a reference to web server 210. Some packets will comprise data that need be supplied to web server 210 (e.g., customer credit information, form data and the like). In these instances, HTTP parser 402 couples to data blender 404.

Optionally, front-end 201 implements security processes, compression processes, encryption processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within any of the functional components (e.g., data blender 404) or implemented as separate functional components within front-end 201. Also, parser 402 may implement a prioritization program to identify packets that should be given higher priority service. A prioritization program requires only that parser 402 include a data structure associating particular clients 205 or particular TCP packet types or contents with a prioritization value. Based on the prioritization value, parser 402 may selectively implement such features as caching, encryption, security, compression and the like to improve performance and/or additional functionality. The prioritization value is provided by the owners of web site 210, for example, and may be dynamically altered, statically set, or updated from time to time to meet the needs of a particular application.

Blender 404 slices and/or coalesces the data portions of the received packets into a more desirable "TMP units" that are sized for transport through the TMP mechanism 212. The data portion of TCP packets may range in size depending on client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where compression is applied, the compressed data will vary in size depending on the compressibility of the data. Data blender 404 receives information from front-end manager 217 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism

212. Data blender 404 also marks the TMP units so that they can be re-assembled at the receiving end.

Data blender 404 may also serve as a buffer for storing packets from all appliances 177 that are associated with front-end 201. In accordance with the present invention, data blender 404 may associate a prioritization value with each packet.

In an exemplary implementation, a "TMP connection" comprises a plurality of "TCP connection buffers", logically arranged in multiple "rings". Each TCP socket maintained between the front-end 201 and a client 205 corresponds to a TCP connection buffer. When a TCP connection buffer is created, it is assigned a priority. For purposes of the present invention, any algorithm or criteria may be used to assign a priority. Each priority ring is associated with a number of TCP connection buffers sockets having similar priority. In a specific example, five priority levels are defined corresponding to five priority rings. Each priority ring is characterized by the number of connection buffers it holds (nSockets), the number of connection buffers it holds that have data waiting to be sent (nReady) and the total number of bytes of data in all the connection buffers that it holds (nBytes).

When composing TMP data packets, the blender goes into a loop comprising the steps:

1) Determine the number of bytes available to be sent from each ring (nBytes), and the number of TCP connections that are ready to send (nReady)

2) Determine how many bytes should be sent from each ring (nSend). This is based on a weight parameter for

each priority. The weight can be thought of as the number of bytes that should be sent at each priority this time through the loop.

3) The nSend value computed in the previous step reflects the weighted proportion that each ring will have in a blended TMP packet, but the values of nSend do not reflect how many bytes need to be selected to actually empty most or all of the data waiting to be sent a single round. To do this, the nSend value is normalized to the ring having the most connections waiting. This involves a calculation of a factor: S = nBytes/(Weight*nReady) for the ring with the greatest nReady. Then, for each ring, calculate nSend X S to get the normalized value (nSendNorm) for each priority ring.

4) Send sub-packets from the different rings. This is done by taking a sub-packet from the highest priority ring and adding it to a TMP packet, then adding a sub-packet from each of the top two queues, then the top three, and so on.

5) Within each ring, sub-packets are selected round robin. When a sub-packet is selected from a TCP connection buffer the ring is rotated so the next sub-packet the ring adds will come from a different TCP connection buffer. Each sub-packet can be up to 512 bytes in a particular example. If the connection buffer has less than 512 bytes waiting, the data available is added to the TMP packet.

6) When a full TMP packet (roughly 1.5 kB in a particular example)is built, it is sent. This will usually have three or more sub-packets, depending on their size. The TMP packet will also be sent when there is no more data ready.

TMP mechanism 405 implements the TMP protocol in accordance with the present invention. TMP is a TCP-like protocol adapted to improve performance for multiple channels operating over a single connection. Front-end TMP mechanism 405 and a corresponding back-end TMP mechanism 505 shown in FIG. 6 are computer processes that implement the end points of TMP link 212. The TMP mechanism in accordance with the present invention creates and maintains a stable connection between two processes for high-speed, reliable, adaptable communication.

TMP is not merely a substitute for the standard TCP environment. TMP is designed to perform particularly well in heterogeneous network environments such as the Internet. TMP connections are made less often than TCP connections. Once a TMP connection is made, it remains up unless there is some kind of direct intervention by an administrator or there is some form of connection breaking network error. This reduces overhead associated with setting up, maintaining and tearing down connections normally associated with TCP.

Another feature of TMP is its ability to channel numerous TCP connections through a single TMP pipe 212. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then mapped into a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 212 in order to traffic the TCP connections to their appropriate destinations. TMP includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

Another advantage of TMP as compared to traditional protocols is the amount of information about the quality

of the connection that a TMP connection conveys from one end to the other of a TMP pipe 212. As often happens in a network environment, each end has a great deal of information about the characteristics of the connection in one direction, but not the other. By knowing about the connection as a whole, TMP can better take advantage of the available bandwidth.

In contrast with conventional TCP mechanisms, the behavior implemented by TMP mechanism 405 is constantly changing. Because TMP obtains bandwidth to host a variable number of TCP connections and because TMP is responsive information about the variable status of the network, the behavior of TMP is preferably continuously variable. One of the primary functions of TMP is being able to act as a conduit for multiple TCP connections. As such, a single TMP connection cannot behave in the same manner as a single TCP connection. For example, imagine that a TMP connection is carrying 100 TCP connections. At this time, it loses one packet (from any one of the TCP connections) and quickly cuts its window size in half (as specified for TCP). This is a performance reduction on 100 connections instead of just on the one that lost the packet.

Each TCP connection that is passed through the TMP connection must get a fair share of the bandwidth, and should not be easily squeezed out. To allow this to happen, every TMP becomes more aggressive in claiming bandwidth as it accelerates. Like TCP, the bandwidth available to a particular TMP connection is measured by its window size (i.e., the number of outstanding TCP packets that have not yet been acknowledged). Bandwidth is increased by increasing the window size, and relinquished by reducing the window size. Up to protocol

specified limits, each time a packet is successfully delivered and acknowledged, the window size is increased until the window size reaches a protocol specified maximum. When a packet is dropped (e.g., no acknowledge received or a resend packet response is received), the bandwidth is decreased by backing off the window size. TMP also ensures that it becomes more and more resistant to backing off (as compared to TCP) with each new TCP connection that it hosts. A TMP should not go down to a window size of less than the number of TCP connections that it is hosting.

In a particular implementation, every time a TCP connection is added to (or removed from) what is being passed through the TMP connection, the TMP connection behavior is altered. It is this adaptation that ensures successful connections using TMP. Through the use of the adaptive algorithms discussed above, TMP is able to adapt the amount of bandwidth that it uses. When a new TCP connection is added to the TMP connection, the TMP connection becomes more aggressive. When a TCP connection is removed from the TMP connection, the TMP connection becomes less aggressive.

TMP pipe 212 provides improved performance in its environment as compared to conventional TCP channels, but it is recognized that TMP pipe 212 resides on the open, shared Internet in the preferred implementations. Hence, TMP must live together with many protocols and share the pipe efficiently in order to allow the other transport mechanisms fair access to the shared communication bandwidth. Since TMP takes only the amount of bandwidth that is appropriate for the number of TCP connections that it is hosting (and since it monitors the connection and controls the number of packets that it puts on the line),

TMP will exist cooperatively with TCP traffic. Furthermore, since TMP does a better job at connection monitoring than TCP and TMP is better suited to throughput and bandwidth management than TCP.

Also shown in FIG. 5 are data filter component 406 and HTTP reassemble component 407 that process incoming (with respect to client 205) data. TMP mechanism 405 receives TMP packets from TMP pipe 212 and extracts the TMP data units. Using the appended sequencing information, the extracted data units are reassembled into HTTP data packet information by HTTP reassembler 407. Data filter component 406 may also implement data decompression where appropriate, decryption, and handle caching when the returning data is of a cacheable type.

FIG. 6 illustrates principle functional components of an exemplary back-end 203 in greater detail. Primary functions of the back-end 203 include translating transmission control protocol (TCP) packets from web server 210 into TMP packets as well as translating TMP packets received from a front-end 201 into the one or more corresponding TCP packets to be send to server 210. Further, back-end 203 is able to implement similar or complementary functionality to that of front-end 203. In this manner, back-end 203 can operate as a web server to retrieve content and generate web pages, analyze and reformat web pages and components within web pages, and similar server functionality that would conventionally be implemented in a server 210. In general, any functionality and behavior described herein that can be implemented on server 210 and/or front-end server 201 can also be implemented on back-end server 203.

TMP unit 505 receives TMP packets from TMP pipe 212 and passes them to HTTP reassemble unit 507 where they are

reassembled into the corresponding TCP packets. Data filter 506 may implement other functionality such as decompression, decryption, and the like to meet the needs of a particular application. The reassembled data is forwarded to TCP component 501 for communication with web server 210.

TCP data generated by the web server process are transmitted to TCP component 501 and forwarded to HTTP parse mechanism 502. Parser 502 operates in a manner analogous to parser 402 shown in FIG. 5 to extract the data portion from the received TCP packets, perform optional compression, encryption and the like, and forward those packets to data blender 504. Data blender 504 operates in a manner akin to data blender 404 shown in FIG. 5 to buffer and prioritize packets in a manner that is efficient for TMP transfer. Priority information is received by, for example, back-end manager 219 based upon criteria established by the web site owner. TMP data is streamed into TMP unit 505 for communication on TMP pipe 212.

FIG. 7 illustrates a conceptual block diagram of an exemplary implementation of the system shown in FIG. 2 in an alternative context. In the example of FIG. 6, front-end 201 is implemented as a front-end web server 601 operating at an ISP 602. ISP 602 supports modem, digital subscriber line (DSL), ISDN, leased line, or other communication ports for communicating with one or more clients 605. ISP 602 serves as a bridge to couple client connections to IP connections with network 101. ISP 602 may be considered to be outside of network 101 in that quality of service between client 605 and ISP 602 is not dependent on congestion or equipment failure or other factors affecting quality of service within network 101.

In operation, client 605 generates an HTTP request specifying web site 610 in the URL. In the manner described hereinbefore, the client request is redirected to front-end 601. Once the redirection is completed, front-end 601 serves web pages embedded in HTTP response packets using both content and functionality obtained from web site 610 as well as content and functionality present on front-end 601 or obtained from database 603. In this manner, front-end web server 601 can dynamically controls the source of the delivered content. The web page(s) served to client 605 comprise a composite of multiple sources from multiple independent web servers.

Significantly, the content served from content database 603 may differ from the content and functionality that would have otherwise been served by web site 610. For example, if web site 610 returns an HTTP 404 "page not found" error page, front-end web site 601 may supply a more informative or instructive web page derived from content database 603. Alternatively, front-end web site 601 may detect periods of low quality of service or slow response and provide substitute content from content database 603 or elsewhere. In a particular example, web site 610 publishes a load index that can be read by front-end 601 and used to generate a wait page intended to occupy a user of client 605 until content can be obtained directly from web site 610.

In another embodiment, the desired format of content is specified by client 605, information maintained about the desired or required format for client 605 by the system and/or the system can determine the required format for client 605 and its connection to front-end 601 indirectly. In response, front-end web server 601 formats and converts content and data to be provided to client 605

based on custom requests forwarded to web server 610 and/or on a common data set provided by website 610 for purposes of responding to one or more clients 605.

In a particular example, the owner of web site 610 establishes rules for how front-end web server 601 handles various conditions. These rules are stored in front-end web server 601. Hence, the owner of web site 610 is not losing control over how web site 610 is presented, but instead is gaining control over how presentation occurs during periods where network 101 is unavailable or provides unacceptable quality of service. By placing a web server in front of the origin web server 610, overall user experience as well as efficacy of the web site 610 for the site owner are improved.

It should be understood that the essence of systems in accordance with the present invention, i.e., dynamically shifting functionality amongst a network of servers can be expressed in a variety of implementations. For example, a network-connected server may obtain program code, data, and other resources to provide a desired set of functionality either in response to a client request, or independently of the client request in a manner that proactively or anticipatorily. Proactive operation allows a network-connected server to maintain a set of functionality such that services can be provided without reference to a central authority when a client request is actually received. In such cases, processes within the front-end 201, for example, will make requests to another network-connected server (e.g., another front-end 201) or to an originating server 210 to obtain the program code, data, and resources, and then independently respond to client requests using the obtained program code, data and resources. The connections between front-ends 201 and

origin servers 210 can benefit from features of the enhanced channel 202.

Any number of front-ends 201 may be involved in providing a particular service or set of functionality. It is contemplated that a client 205 may connect to a particular front-end 201 and that particular front-end 201 may establish connections with one or more other front-ends 201 to access functionality, data and resources available in those other front-ends.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols and languages including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols and languages such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end supports a DBMS protocol such as SQL. Such implementations not only provide the advantages of the present invention, but also enable a client to access a rich set of network resources with minimal client software.